Subquadratic Space Complexity Binary Field Multiplier Using Double Polynomial Representation

Jean-Claude Bajard, Christophe Negre, and Thomas Plantard

Abstract—This paper deals with binary field multiplication. We use the bivariate representation of binary field called Double Polynomial System (DPS) presented in [11]. This concept generalizes the composite field representation to every finite field. As shown in [11], the main interest of DPS representation is that it enables to use Lagrange approach for multiplication, and in the best case, Fast Fourier Transform approach, which optimizes Lagrange approach. We use here a different strategy from [11] to perform reduction, and we also propose in this paper, some new approaches for constructing DPS. We focus on DPS, which provides a simpler and more efficient method for coefficient reduction. This enables us to avoid a multiplication required in the Montgomery reduction approach of [11], and thus to improve the complexity of the DPS multiplier. The resulting algorithm proposed in the present paper is subquadratic in space $O(n^{1.31})$ and logarithmic in time. The space complexity is 33 percent better than in [11] and 18 percent faster. It is asymptotically more efficient than the best known method [6] (specifiably more efficient than [6] when $n \geq 3{,}000$). Furthermore, our proposal is available for every n and not only for n a power of two or three.

Index Terms—Binary field, double polynomial system, multiplication, subquadratic complexity, FFT.

1 Introduction

Efficient finite field arithmetic in $\mathbb{F}_{2^n}$ is one of the challenges in implementing cryptographic cryptosystem like elliptic curve cryptography or cryptosystems based on DLP in finite fields. A binary field $\mathbb{F}_{2^n}$ can be seen as the set of binary polynomials with degree $<n$. Multiplication and addition in $\mathbb{F}_{2^n}$ are done modulo a degree n irreducible polynomial P.

In order to get efficient reduction modulo P, NIST recommends [3], [15] to use P with trinomial (or pentanomials if there are no irreducible trinomials of degree n) form. In this case, the architecture is dedicated to only one P, which is not fine for circuit makers. In this paper, the approach proposed is available for every P.

There are two types of binary field multipliers. The first ones are called sequential multipliers, their hardware space complexity is $O(n)$, and their critical path have a delay of $O(1)$ or $O(\log(n))$ (see, for example, [24], [25], a complete multiplication is done after n clock cycles using the same hardware, thus the time complexity is in $O(n)$ or $O(n\log(n))$. The second kind of multipliers are the parallel multipliers, they are faster: their time complexity is $O(\log(n))$, but their space complexity is for the best one subquadratic $O(n^{1+\epsilon})$ [6]. The approach proposed in this paper belongs to this second category, and is asymptotically better that the former ones found in the literature.

- *J.C. Bajard is with UPMC Paris, LIP6 CNRS, France.*
- *C. Negre is with DALI/ELIAUS, Université de Perpignan, France.*
- *T. Plantard is with the University of Wollongong, Australia.*

Manuscript received 14 Dec. 2008; revised 16 Oct. 2009; accepted 27 Jan. 2010; published online 10 June 2010.
Recommended for acceptance by P. Montuschi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-12-0615.
Digital Object Identifier no. 10.1109/TC.2010.141.

1.1 State of the Art on Parallel Multiplier

In his PhD [19], Mastrovito expresses the finite field $\mathbb{F}_{2^n}$ multiplication as a product of an $n \times n$ matrix by a vector. He gives an algorithm for constructing this $n \times n$ matrix, and shows that trinomials and pentanomials offer efficient implementations. This approach is used in many other works concerning multiplication in $\mathbb{F}_{2^n}$ [14], [13]. This strategy is improved by Doche in [4]. He proposes to use redundant trinomials, when the field cannot be defined by an irreducible trinomial.

Fan and Hasan [6] compute the product of two field elements using a matrix-vector product as proposed by Mastrovito. They propose to use a divide and conquer approach to perform efficiently this matrix-vector product. Their method is available for n such that n is a power of two or three.

As the square operation appears in the addition formulas of two points of an elliptic curve,[1] Massey and Omura [18] proposed to represent the field in a normal basis. In this case, the evaluation of the square is reduced to a cyclic shift, but the multiplication (in an arbitrary normal basis) can be very costly. In [21], Mullin et al. show that some normal bases can be optimal for the multiplication. But optimal normal bases do not exist for every field $\mathbb{F}_{2^n}$. For such a case, the best known method has been proposed by Fan and Hasan. In [7], they successfully adapt their method [6] initially proposed for polynomial representation to optimal normal basis.

The Fan and Hasan's divide and conquer approach [6], [7] provides a multiplier for n a power of two or three with $O(n^{1.58})$ gates for the space complexity and $O(\log(n))$ in time.

A new system of representation called the Double Polynomial System (DPS) is introduced in [16] and [11]. It is a variant of the polynomial representation inspired from the adapted bases for the modular arithmetic [2]. They

1. The exact formula can be found, for example, in [3].

represent the elements in a double polynomial representation: the field elements are polynomials in two variables α and β with bounded degrees. In [11], they provide a field multiplier based on a Montgomery method [20]. One interest of their system of representation is that it allows the use of Fast Fourier Transform (FFT) for polynomial multiplication. The use of FFT is a classical and efficient approach for polynomial multiplication [9]: this consists in evaluating the polynomial in roots of unity so as to perform the multiplication in this Lagrange representation and to finally get the product using an interpolation.

Until now, this is the best known method. Its space complexity is roughly $85n^{1.31}$ gates (XOR or AND) and its time complexity is $16\log_3(n)T_X + T_A$, where T_X and T_A represent, respectively, the delay of an XOR gate and an AND gate.

1.2 Our Results

In this paper, we focus on some specific DPS, so as to avoid the use of a Montgomery approach presented in [11] for reduction. We provide a new method to perform reduction in these specific DPSs, which is more efficient than Montgomery approach. We give some original methods for constructing these DPSs. Following the idea found in [11], we propose a version of our algorithm using an FFT mixing DPS and Lagrange representation. Compared to [11], we avoid several FFT computation and coefficient multiplications, using the features of our new point of view. Our resulting multiplier has a space complexity of $\cong 42n^{1.31}$ gates and a delay of $13\log_3(n)T_X$. Moreover, we propose, in Section 7.1, an efficient squarer that was not provided in [11].

2 The Double Polynomial System

One of the most commonly used methods to represent elements of a binary field consists in using a generating system $\mathcal{G}$ of $\mathbb{F}_{2^n}$ (we note that it is not necessarily a basis). Each field element is expressed as a sum of the elements of $\mathcal{G}$, and we represent this element by its coordinates in the system.

Definition 1 (Generating system). *A set $\mathcal{G} = (\beta_1, \ldots, \beta_k)$ of k elements of $\mathbb{F}_{2^n}$, with $k \geq n$, is a generating system if every element U of $\mathbb{F}_{2^n}$ can be written as:*

$$U = \sum_{i=1}^{k} u_i \beta_i, \qquad \text{with } u_i \in \{0,1\}. \tag{1}$$

For each element $U \in \mathbb{F}_{2^n}$, the vector $(u_1, \ldots, u_k)_{\mathcal{G}}$ represents the coordinates of U in $\mathcal{G}$ and is called the representation of U in $\mathcal{G}$. If $k = n$, then the generating system is a basis of $\mathbb{F}_{2^n}$, and for each $U \in \mathbb{F}_{2^n}$, the representation of $U = (u_1, \ldots, u_n)_{\mathcal{G}}$ is unique.

In the following, we simplify, by often omitting the subscript $(\cdot)_{\mathcal{G}}$. In a generating system representation, the addition of two elements $U, V \in \mathbb{F}_{2^n}$ is just a bitwise XOR of the element coordinates. The multiplication is a little bit more complex. For each field $\mathbb{F}_{2^n}$, we have to choose the best generating system to obtain the most efficient implementation of the multiplication.

2.1 Usual Representation System

The following two systems are the most commonly used for representing binary fields:

- *Polynomial bases* are bases of $\mathbb{F}_{2^n}$ of the following form:

  $$\mathcal{B} = (1, \beta, \beta^2, \ldots, \beta^{n-1}),$$

  such that $\beta \in \mathbb{F}_{2^n}$ has a minimal polynomial of degree n. Mastrovito, in his thesis [19], showed that these bases are particularly interesting when the minimal polynomial of β is sparse (e.g., trinomial or pentanomial). Indeed, in these cases, the reduction modulo the irreducible polynomial of β is really simple.
- *The normal bases* are the bases of $\mathbb{F}_{2^n}$ of the following form:

  $$\mathcal{B} = (\zeta, \zeta^2, \zeta^4, \ldots, \zeta^{2^{n-1}}).$$

  Clearly, the elements ζ^{2^i} must be linearly independent. These bases yield a very simple way for squaring the elements of $\mathbb{F}_{2^n}$: this is done by a cyclic shift of the coefficients. For general normal bases, the multiplication is not really efficient. Vanstone and coworkers [21] proposed a special family of normal bases, i.e., so-called optimal normal bases (ONB), which provide efficient multiplication in $\mathbb{F}_{2^n}$.

The notion of dual basis [8] is sometimes used for constructing a multiplier. Generally, such approaches are interesting when the dual basis is constructed over polynomial bases modulo sparse irreducible polynomials or over an optimal normal base.

In [11], the authors introduce a new generating system: the Double Polynomial System.

Definition 2 (DPS [11]). *We call Double Polynomial System of $\mathbb{F}_{2^n}$, a generating system defined by five elements (α, r, β, m, p), where $\alpha, \beta \in \mathbb{F}_{2^n}$ and $m, r \in \mathbb{N}$, and p is an irreducible polynomial which defines $\mathbb{F}_{2^n}$, such that each element $U \in \mathbb{F}_{2^n}$ can be written as*

$$U = \sum_{i=0}^{m-1} \sum_{j=0}^{r-1} u_{i,j} \alpha^j \beta^i \pmod p, \quad \textit{with } u_{i,j} \in \{0,1\},$$

or, similarly,

$$U = \sum_{i=0}^{m-1} u_i(\alpha) \beta^i \pmod p \quad \textit{with } \forall i, \deg_\alpha u_i(\alpha) < r.$$

Remark 1. We note that the family $\alpha^j \beta^i$, for $i = 0, \ldots, m-1$ and $j = 0, .., r-1$ form a generating system $\mathcal{G}$ with $k = m \times r$.

In the following examples, we give two different ways to check that the DPS is indeed a generating system:

Example 1. Let $\mathbb{F}_{2^5} = \mathbb{F}_2[X]/p(X)$ with $p = X^5 + X^3 + X^2 + X + 1$, and let $\alpha = X, \beta = 1 + X^3 + X^4$. Then, the system $\mathcal{S} = (\alpha, 3, \beta, 2, p)$ is a DPS. It is indeed equivalent to the following generating system:

$$\begin{aligned}\mathcal{G} &= (1, \alpha, \alpha^2, \beta, \alpha\beta, \alpha^2\beta) \\ &= (1, X, X^2, 1 + X^3 + X^4, 1 + X^2 + X^3 + X^4, 1 + X^2 + X^4).\end{aligned}$$

Let us verify that $\mathcal{G}$ generates $\mathbb{F}_{2^5}$. We have to express an element $U \in \mathbb{F}_{2^n}$ in the system $\mathcal{S}$. We begin from the

following expression of U in the polynomial basis $(1, X, X^2, X^3, X^4)$:

$$U = \tilde{u}_0 + \tilde{u}_1 X + \tilde{u}_2 X^2 + \tilde{u}_3 X^3 + \tilde{u}_4 X^4, \quad (2)$$
$$U = \tilde{u}_0 + \tilde{u}_1 \alpha + \tilde{u}_2 \alpha^2 + \tilde{u}_3 \alpha^3 + \tilde{u}_4 \alpha^4. \quad (3)$$

We can check that α^3 and α^4 can be written as follows:

$$\alpha^4 = \alpha^2\beta + \alpha^2 + 1,$$
$$\alpha^3 = \alpha^2 + \beta + \alpha^2\beta.$$

Thus, in (2), by replacing α^4 and α^3 by the previous expressions, we get:

$$\begin{aligned} U &= (\tilde{u}_0 + \tilde{u}_4) + \tilde{u}_1\alpha + (\tilde{u}_2 + \tilde{u}_3 + \tilde{u}_4)\alpha^2 + (\tilde{u}_3)\beta \\ &\quad + (0)\alpha\beta + (\tilde{u}_3 + \tilde{u}_4)\alpha^2\beta \\ &= u_{0,0} + u_{0,1}\alpha + u_{0,2}\alpha^2 + u_{1,0}\beta + u_{1,1}\alpha\beta + u_{1,2}\alpha^2\beta \\ &= u_0(\alpha) + u_1(\alpha)\beta, \end{aligned}$$

where $u_0(\alpha) = u_{0,0} + u_{0,1}\alpha + u_{0,2}\alpha^2$ and $u_1(\alpha) = u_{1,0} + u_{1,1}\alpha + u_{1,2}\alpha^2$, which proves that $\mathcal{S}$ is a generating system.

Example 2. We use the same field as in Example 1. We consider $\alpha = 1 + X$ and $\beta = 1 + X^3 + X^4$. These two elements satisfy the following equations:

$$\begin{aligned} X^4 &= \alpha^2\beta + \alpha^2 + \beta \mod p, \\ X^3 &= 1 + \alpha^2 + \alpha^2\beta \mod p, \\ X^2 &= \alpha^2 + 1 \mod p, \\ X^1 &= \alpha + 1 \mod p, \\ X^0 &= 1. \end{aligned}$$

Every $U = \sum_{i=0}^{4} u_i X^i \in \mathbb{F}_{2^5}$ can be expressed in the system $\mathcal{S} = (\alpha, 3, \beta, 2, p)$ by replacing each X^i in U by its corresponding expression in $\mathcal{S}$. The corresponding generating system is

$$\begin{aligned} \mathcal{G} &= (1, \alpha, \alpha^2, \beta, \alpha\beta, \alpha^2\beta) \\ &= (1, 1 + X, 1 + X^2, 1 + X^3 + X^4, X^2, X^2 + X^3). \end{aligned}$$

At the end of the 1990s, several implementations for composite fields $\mathbb{F}_{2^{m\times r}}$ were proposed by Paar and coworkers [12] and DeWin et al. [5]. They used the fact that $\mathbb{F}_{2^{m\times r}}$ is a field extension of $\mathbb{F}_{2^r}$ of degree m. They represent $\mathbb{F}_{2^r}$ with a polynomial basis $(\alpha^i)_{i=0}^{r-1}$ over $\mathbb{F}_2$ and they represented $\mathbb{F}_{2^{r\times m}}$ using a polynomial basis $(\beta^i)_{i=0}^{m-1}$ over $\mathbb{F}_{2^r}$. The elements of $\mathbb{F}_{2^{m\times r}}$ are, in this situation, polynomials in two variables α and β. The double polynomial system generalizes this kind of representation for noncomposite fields $\mathbb{F}_{2^n}$, i.e., with n prime. In this case, a DPS is always redundant, because $m \times r$ must be strictly bigger than n, when n is prime, to be sure that the representation is sufficiently large to represent the field. In Section 3, we propose a DPS class where the reduction and multiplication are more efficient than in the classical representations.

Remark 2. We note that, in this paper, we do not consider elements of the field $\mathbb{F}_{2^{mr}}$, but elements of $\mathbb{F}_{2^n}$ coded on $m \times r$ bits, with $n < m \times r$.

3 Adapted DPS for Multiplication in $\mathbb{F}_{2^n}$

This section deals with the multiplication in a DPS of $\mathbb{F}_{2^n}$. We propose to use a classical approach, where the multiplication is decomposed in two steps: first, a polynomial multiplication, and then, a modular reduction. We consider two elements of $\mathbb{F}_{2^n}$, $U = \sum_{i=0}^{m-1} u_i(\alpha)\beta^i$ and $V = \sum_{i=0}^{m-1} v_i(\alpha)\beta^i$, which are expressed in a double polynomial system $\mathcal{S} = (\alpha, r, \beta, m, p)$. The product W of U and V can be expressed as:

$$\begin{aligned} W = UV &= \left(\sum_{i=0}^{m-1}\sum_{j=0}^{r-1} u_{i,j}\alpha^j\beta^i\right)\left(\sum_{i=0}^{m-1}\sum_{j=0}^{r-1} v_{i,j}\alpha^j\beta^i\right) \\ &= \sum_{i=0}^{2m-2}\sum_{j=0}^{2r-2} w_{i,j}\alpha^j\beta^i. \end{aligned}$$

In this expression, some terms $\alpha^i\beta^j$ are such that $i \geq m$ or/and $j \geq r$. These terms must be reduced to obtain a representation of this value in the DPS $\mathcal{S}$.

In [11], they define a specific kind of DPS, called *Adapted* DPS (ADPS) which provides simple reduction in β. The reduction in α will be considered in Section 3.2.

Definition 3 (Adapted Double Polynomial Systems [11]). *Let (α, r, β, m, p) be a DPS, we say that $(\alpha, r, \beta, m, c, p)$ is an Adapted DPS if α and β verify:*

$$\beta^m = c(\alpha) \pmod p,$$

with $c \in \mathbb{F}_2[\alpha]$ and $\deg_\alpha c(\alpha)$ very small.

In an ADPS, the reduction of the degree in β of W is an easy process. Hence, the multiplication in an ADPS is decomposed in three steps: first, we consider the ADPS representations as polynomials in β with coefficients in $\mathbb{F}_2[\alpha]$, and we multiply these polynomials. Then, we have two further reduction steps: a first one for reducing this product modulo $(\beta^m - c(\alpha))$ as a polynomial in β, and a second one for reducing its coefficients, which are polynomials in α, to a degree lower than r.

The multiplication algorithm is depicted below:

Algorithm 1. Multiplication in an ADPS (ADPS_Multiplication)
Require: One ADPS $(\alpha, r, \beta, m, c, p)$ and two elements of $\mathbb{F}_{2^n}$, $U = (u_0(\alpha), \ldots, u_{m-1}(\alpha))$ and $V = (v_0(\alpha), \ldots, v_{m-1}(\alpha))$.
Ensure: W.
Polynomial multiplication in $(\mathbb{F}_2[\alpha])[\beta]$. $A(\beta) \leftarrow U(\beta)V(\beta)$
Polynomial reduction. $B(\beta) \leftarrow A(\beta) \bmod (\beta^m - c(\alpha))$ (reduction in β)
Coefficients reduction. $W(\beta) \leftarrow CR(B(\beta))$ (reduction in α)

The two first steps of this algorithm are classical, and we depict them in a short section where we give the expression of the obtained polynomials with their degrees in α. We will present later in Sections 5 and 6 an efficient hardware architecture which performs these two operations at the same time, using FFT.

We focus on the "Coefficient reduction" which is the original part of the algorithm. We show that if there exists a sparse ADPS representation of α^r, it is possible to have an efficient coefficient reduction.

3.1 Analysis of the First Two Steps

3.1.1 Multiplication of Polynomials in β

The polynomial A is the product of U and V considered as polynomials in β.

The obtained polynomial A is such that:

$$A(\beta) = UV = \left(\sum_{i=0}^{m-1} u_i(\alpha)\beta^i\right)\left(\sum_{j=0}^{m-1} v_j(\alpha)\beta^j\right) = \sum_{i=0}^{2m-2} a_i(\alpha)\beta^i,$$

where the $a_i(\alpha)$ are polynomials in $\alpha : a_i(\alpha) = \sum_{k=0}^{i} u_k(\alpha)v_{i-k}(\alpha)$.

Thus, the maximal degree in α of the coefficient of A is written as:

$$\deg_\alpha A = \max_{0\le i\le 2m-2}\left(\max_{0\le k\le i}(\deg_\alpha u_k(\alpha) + \deg_\alpha v_{i-k}(\alpha))\right) \le 2r-2. \quad (4)$$

We thus assume that the degree in α of these coefficients $a_i(\alpha)$ is smaller than or equal to $2r-2$.

3.1.2 Polynomial Reduction in β.

In the second step, the previous result A is reduced modulo $\beta^m - c(\alpha)$. For this, we decompose A as a polynomial in β, in two parts, one of degree lower than m, and one larger than or equal to m:

$$A = \sum_{i=0}^{m-1} a_i(\alpha)\beta^i + \beta^m \sum_{i=0}^{m-2} a_{m+i}(\alpha)\beta^i.$$

Thus, we obtain $B(\beta)$, which is equal to $A(\beta) \bmod (\beta^m - c(\alpha))$, by replacing β^m by $c(\alpha)$:

$$B = \sum_{i=0}^{m-1} a_i(\alpha)\beta^i + c(\alpha) \sum_{i=0}^{m-2} a_{m+i}(\alpha)\beta^i.$$

If we note $a_{2m-1} = 0$, we get:

$$B = \sum_{i=0}^{m-1} b_i(\alpha)\beta^i \quad \text{where } b_i(\alpha) = a_i(\alpha) + c(\alpha)a_{i+m}(\alpha). \quad (5)$$

Now, we evaluate the maximal degree in α of the coefficients of B in (5). Equations (5) and (4) imply that, for $i = 0, \ldots, m-1$, the maximal degree in α of the coefficients of B satisfies:

$$\deg_\alpha B = \max_{i=0}^{m-1}(\deg_\alpha b_i(\alpha)) \le 2r - 2 + \deg_\alpha c(\alpha). \quad (6)$$

Hence, we must reduce the coefficients of B to a degree smaller than r in the representation obtained in (5) for obtaining an ADPS representation W equivalent to B. Coefficients of B are considered as polynomials in α whose degree must be reduced using some properties of the considered ADPS. Thus, at the end of the algorithm of coefficients reduction, we get an expression of W with a degree in α lower than r.

Remark 3. For the multiplication step, different approaches can be available depending on the size of m and r: for example, Karatsuba or Toom-Cook schemes. Then, the complexity of the polynomial reduction in β is related to the Hamming weight of $c(\alpha)$. But, we will perform these two steps using a Fast Fourier Transform approach in Section 5.

3.2 Coefficient Reduction

Coefficients $b_i(\alpha)$ of B are considered as polynomials in α. We know that their degrees are smaller than or equal to $2r - 2 + \deg_\alpha c(\alpha)$. To obtain an ADPS representation, we must reduce them to a degree lower than r.

To achieve this goal, we propose to consider Z an ADPS representation of α^r:

$$\alpha^r = Z = z_0(\alpha) + z_1(\alpha)\beta + \cdots + z_{m-1}(\alpha)\beta^{m-1}, \quad (7)$$

where $\deg_\alpha z_i(\alpha) < r$.

The reduction process consists in replacing α^r by Z several times and smartly in the expressions of $b_i(\alpha)$. We first deal with a special case which works on polynomials B having a small degree $(r + \Delta)$ in α called a *semireduction process* (Δ depends on the degree of Z, see Theorem 1). After that, we will deal with the case of a general B.

3.2.1 The Semireduction Process.

This process is the basic keystone of the coefficient reduction (Algorithm 3). In this part, we consider B as a polynomial in β with coefficients in $\mathbb{F}_2[\alpha]$ of degree lower than or equal to $r + \Delta$. The output of the semireduction will be an equivalent polynomial for the ADPS, with coefficients in $\mathbb{F}_2[\alpha]$ of degree lower than or equal to $r - 1$. We note $\Delta + 1$ to be the part of the degree which will be reduced in Algorithm 2. Then, the semireduction process constructs from a polynomial B with coefficients of degree in α smaller than or equal to $r + \Delta$, an equivalent polynomial (ADPS mean) with coefficients of degree smaller than or equal to $r - 1$. The term Δ depends on Z, i.e., the representation (7) of α^r in the ADPS, and on the degree of $c(\alpha)$.

Algorithm 2 uses representation Z for replacing the multiplication with α^r by one matrix-vector product (that we will reduce to few additions) by the low coefficient matrix M defined in the proof of Theorem 1. Hence, this algorithm computes an expression of B with a degree in α smaller than r, i.e., an expression of B in the considered ADPS.

Algorithm 2. $SR(R, \mathcal{S})$, semireduction process

Require: An ADPS $\mathcal{S} = (\alpha, r, \beta, m, c(\alpha), p)$ of a finite field $\mathbb{F}_{2^n}$, $\alpha^r = (z_0(\alpha), \ldots, z_{m-1}(\alpha))_{\mathcal{S}}$ with $Z = \sum_{i=0}^{m-1} z_i(\alpha)\beta^i$ an expression of α^r in the ADPS, a matrix M defined by (12), and a vector $B = (b_0(\alpha), \ldots, b_{m-1}(\alpha))_{\mathcal{S}}$ with $b_i \in \mathbb{F}_2[\alpha]$ and a degree $\le \Delta + r$.

Ensure: B semireduced.

Define the vectors $\underline{B}$ and $\overline{B}$ such that $B = \underline{B} + \alpha^r\overline{B}$ and $\deg_\alpha \underline{B} < r$

Compute $B \leftarrow \underline{B} + M \cdot \overline{B}$.

Return B

Theorem 1. *If* $\Delta = (r - 1 - \deg_\alpha Z - \deg_\alpha c(\alpha))$, *with* $\deg_\alpha Z = \max_{i=0}^{m-1} \deg_\alpha z_i(\alpha)$, *then Algorithm 2 constructs from a polynomial B of degree smaller or equal to $r + \Delta$ in α, an equivalent polynomial (representing the same element of $\mathbb{F}_{2^n}$) of degree smaller than or equal to $r - 1$.*

Proof. Let us consider an element $B = \sum_{i=0}^{m-1} b_i(\alpha)\beta^i$ such that coefficients $b_i(\alpha)$ have a degree lower than or equal to $r + \Delta$. We split B into two parts $\underline{B}$ and $\overline{B}$, with respect to the degree in α of its coefficients

$$B = \underline{B} + \alpha^r \overline{B}, \tag{8}$$

where

$$\underline{B} = \sum_{i=0}^{m-1} b'_i(\alpha)\beta^i \quad \text{with } \deg_\alpha b'_i(\alpha) \le r-1,$$
$$\overline{B} = \sum_{i=0}^{m-1} b''_i(\alpha)\beta^i \quad \text{with } \deg_\alpha b''_i(\alpha) \le \Delta.$$

The term $\underline{B}$ is a polynomial verifying the features of an ADPS representation and the term $\overline{B}$ satisfies $\deg_\alpha \overline{B} \le \Delta$. We consider the second term of (8): the product $\alpha^r\overline{B}$, which gives the maximal degree in α in the expression (8) of B. We are going to expand the product $\alpha^r\overline{B}$ to get an expression with degree in α smaller than or equal to $r-1$.

The product $\alpha^r\overline{B}$ is evaluated, first by replacing $\overline{B}$ by this expression. We have

$$\alpha^r\overline{B} = \sum_{i=0}^{m-1} b''_i(\alpha)\alpha^r\beta^i. \tag{9}$$

Now, we replace each $\alpha^r\beta^i$ by an expression in α and β with degree in α smaller than r. We get these expressions by replacing α^r by the expression of Z given in (7), and then, by evaluating the products $Z\beta^i$ modulo $\beta^m - c(\alpha)$:

$$\begin{aligned}
\beta\alpha^r &= c(\alpha)z_{m-1}(\alpha) + z_0(\alpha)\beta + z_1(\alpha)\beta^2 + \cdots \\
&\quad \cdots + z_{m-2}(\alpha)\beta^{m-1}, \\
\beta^2\alpha^r &= c(\alpha)z_{m-2}(\alpha) + c(\alpha)z_{m-1}(\alpha)\beta + z_0\beta^2 + \cdots \\
&\quad \cdots + z_{m-3}(\alpha)\beta^{m-1}, \\
&\vdots \qquad\qquad\qquad \vdots \\
\beta^{m-1}\alpha^r &= c(\alpha)z_1(\alpha) + c(\alpha)z_2(\alpha)\beta + \cdots \\
&\quad \cdots + c(\alpha)z_{m-1}(\alpha)\beta^{m-2} + z_0(\alpha)\beta^{m-1}.
\end{aligned} \tag{10}$$

The expression of $\alpha^r\overline{B}$ can be computed with a matrix-vector product

$$\alpha^r\overline{B} = Z \times \overline{B} = M \cdot \overline{B}, \tag{11}$$

where $\overline{B}$ is considered as a vector and M is the $m \times m$ matrix whose columns are equal to the coefficients of $\alpha^r\beta^i$ in (10)

$$M = \begin{bmatrix} z_0(\alpha) & c(\alpha)z_{m-1}(\alpha) & \cdots & c(\alpha)z_1(\alpha) \\ z_1(\alpha) & z_0(\alpha) & \cdots & c(\alpha)z_2(\alpha) \\ \vdots & & & \vdots \\ z_{m-1}(\alpha) & z_{m-2}(\alpha) & \cdots & z_0(\alpha) \end{bmatrix}. \tag{12}$$

The maximal degree in α of (11) satisfies:

$$\begin{aligned}
\deg_\alpha(M \cdot \overline{B}) &\le \max_{i,j}\Big(\deg_\alpha\Big(c(\alpha)z_i(\alpha)b''_j(\alpha)\Big)\Big) \\
&\le \deg_\alpha(c(\alpha)) + \deg_\alpha(Z) + \Delta,
\end{aligned} \tag{13}$$

since $\deg_\alpha(z_i(\alpha)) \le \deg_\alpha(Z)$ and $\deg_\alpha(b''_j(\alpha)) \le \Delta$. Now, using the fact that $\Delta = r - 1 - \deg_\alpha(Z) - \deg_\alpha(c(\alpha))$, we get

$$\deg_\alpha(M \cdot \overline{B}) \le r - 1.$$

This means that $\alpha^r\overline{B} = M \cdot \overline{B}$ and $\underline{B} + M \cdot \overline{B}$ are both expressed in the ADPS $\mathcal{S} = (\alpha, r, \beta, m, c(\alpha), p)$. □

3.2.2 General Coefficient Reduction Process

The full reduction of B works as follows: we iteratively apply the SR algorithm to the upper part Q of degree $r + \Delta$. B is split into polynomials R and Q such that $B = R + \alpha^t Q$, where the degree in α of R is smaller than or equal to $t - 1$, and that of Q is equal to $r + \Delta$. We have $\deg_\alpha B = t + r + \Delta$.

The degree of B decreases at each step (to $t + r - 1$). After a sufficient number of semireductions, we obtain an ADPS representation of B. The following lemma gives an upper bound on the necessary number of calls to Algorithm 2 for a complete reduction to an ADPS representation:

Lemma 1. *Let $\mathcal{S} = (\alpha, r, \beta, m, c(\alpha), p)$ be an ADPS of a finite field $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/(p)$ and let Z be the expression of α^r in the ADPS (7). We denote by $B = (b_0(\alpha), \ldots, b_{m-1}(\alpha))$ an element of $\mathbb{F}_2[\alpha]^m$, and we set $\Delta = r - 1 - \deg_\alpha(Z) - \deg_\alpha c(\alpha)$.*

$$\text{If } \Delta \ge 0, \textit{i.e., } \textit{if } r > \deg_\alpha c(\alpha) + \deg_\alpha(Z) + 1,$$

then the number N of calls of the semireduction Algorithm 2 to obtain an ADPS expression of B is bounded by

$$N \le \left\lceil \frac{\deg_\alpha B - (r-1)}{\Delta + 1} \right\rceil. \tag{14}$$

Proof. We decompose B as $B = R + \alpha^t Q$, where $t = \deg_\alpha B - (r + \Delta)$ is such that $\deg_\alpha R < t$ and $\deg_\alpha Q = r + \Delta$. Let us show that the reduced value of B, $R + \alpha^t SR(Q)$, is equal to B modulo p. We have seen previously that $SR(Q) = Q \bmod p$. This implies that $R + \alpha^t SR(Q) = R + \alpha^t Q = B \bmod p$.

We consider the degree of $(R + \alpha^t SR(Q))$. The degree of $SR(Q)$ is smaller than r, thus we have

$$\deg_\alpha(R + \alpha^t SR(Q)) \le t + r - 1 = \deg_\alpha B - (\Delta + 1).$$

Consequently, at each call of Algorithm 2, the degree of B decreases by $(\Delta + 1) > 0$ and the value modulo p of B remains unchanged. After ℓ calls for semireduction, we obtain a reduced expression B_ℓ of B such that:

$$\deg_\alpha B_\ell \le \deg_\alpha B - \ell(\Delta + 1).$$

This means that the number N of calls of Algorithm 2 satisfies

$$N \le \left\lceil \frac{\deg_\alpha B - (r-1)}{\Delta + 1} \right\rceil.$$

□

Corollary 1. *If we consider B as the output of the first two steps of Algorithm 1, then the maximal degree of B is given by (6): $\deg_\alpha B \le 2r - 2 + \deg_\alpha c(\alpha)$, and we obtain:*

$$N \leq \left\lceil \frac{r-1+\deg_\alpha c(\alpha)}{r-\deg_\alpha Z-\deg_\alpha c(\alpha)} \right\rceil. \quad (15)$$

In the following table, we give the corresponding upper bound corresponding to (15) in the specific situation $\deg_\alpha(Z)=0$ (this is the case in practical situations). We remark that N remains small even for quite a big value of $\deg_\alpha(c)$.

range of $\deg_\alpha(c)$	N
{0}	1
{1,2,...,r/3}	2
{r/3+1,r/3+2,...,r/2}	3
{r/2+1,r/3+2,...,3r/5}	4

According to Lemma 1, as $(\Delta+1)$ increases, the speed of the coefficient reduction process increases, i.e., when the degrees in α of $c(\alpha)$ and Z decrease. Indeed, in this case, the right part of (14) is smaller. In Section 4, we will study some special ADPS for which $c(\alpha)$ and Z have a low degree in α.

Now, we introduce the full algorithm for coefficient reduction.

Algorithm 3. $CR(B)$, Coefficient Reduction
Require: An ADPS $\mathcal{S}=(\alpha,r,\beta,m,c(\alpha),p)$ of a finite field $\mathbb{F}_{2^n}=\mathbb{F}_2[X]/(p(X))$, the expression $Z=(z_0,\ldots,z_{m-1})$ of α^r in the system $\mathcal{S}$, such that the degrees in α of Z and $c(\alpha)$ satisfy $\Delta=r-1-\deg_\alpha Z-\deg_\alpha c(\alpha)\geq 0$, and a vector $B=(b_0,\ldots,b_{m-1})$ with entries in $\mathbb{F}_2[\alpha]$ (which represents the coefficients of the polynomial B in β).
Ensure: W the reduced expression in α of B.
$W \leftarrow B$
$k \leftarrow \deg_\alpha W$
while $k \geq r$ **do**
$t \leftarrow \max(k-(r+\Delta),0)$
We define Q and R such that: $W=Q\alpha^t+R$
$W \leftarrow SR(Q)\alpha^t+R$
$k \leftarrow \deg_\alpha W$
end while

4 Construction of ADPS

For practical use of Algorithm 1, we need ADPS with sparse $c(\alpha)$ and sparse $z_i(\alpha)$. Indeed, the first step of Algorithm 1 is a classical polynomial multiplication, and the two steps of reduction depend on the Hamming weight of $c(\alpha)$ and $z_i(\alpha)$.

In this section, we present two methods for constructing an ADPS of $\mathbb{F}_{2^n}$ which satisfies these conditions. Our methods are consequences of the following result:

Lemma 2. *Let $\mathbb{F}_{2^n}=\mathbb{F}[X]/(p(X))$ and $\alpha,\beta\in\mathbb{F}_{2^n}$ which satisfy*

$$\begin{aligned}\beta^m &= c(\alpha),\\ \alpha^r &= \sum_{i=0}^{m-1} z_i(\alpha)\beta^i.\end{aligned} \quad (16)$$

If X can be expressed as

$$X=\sum_{i=0}^{r-1}\sum_{j=0}^{m-1} x_i\alpha^i\beta^j, \quad \text{with } x_i\in\{0,1\},$$

and if $\Delta=r-1-\deg_\alpha(Z)-\deg_\alpha c(\alpha)>0$, then the system $\mathcal{G}=(\alpha^i\beta^j)_{0\leq i<r,0\leq j<m}$ is a generating system of $\mathbb{F}_{2^n}$.

Proof. We have to show that each element of $\mathbb{F}_{2^n}=\mathbb{F}[X]/(p(X))$ admits a representation in $\mathcal{G}$. Let us first show that X^i for $i\geq 0$ can be expressed in $\mathcal{G}$. We prove it by induction on i. This is clearly true for $i=0$ and 1. Suppose it is true for i and let us show it for $i+1$. Since $X^{i+1}=X^iX$ and $\mathcal{S}=(\alpha,r,\beta,m,c,p)$ satisfy the condition of Lemma 1, we can apply Algorithm 1 to compute this product. The result is a representation of X^{i+1} in $\mathcal{G}$.

Let U be an arbitrary element of $\mathbb{F}_{2^n}$. By construction of $\mathbb{F}_{2^n}$, U can be expressed as

$$U=\sum_{i=0}^{n-1} u_iX^i. \quad (17)$$

Now, if we replace in (17) each X^i by its corresponding representation in $\mathcal{G}$, we get the required representation of U in $\mathcal{G}$. □

We now give two methods to construct such $p(X)$, α, β. The first method focuses on the case $\beta=X$, i.e., p is the minimal polynomial of β. The second one deals with $\alpha=X$, i.e., p is the minimal polynomial of α.

4.1 Construction of the Minimal Polynomial of β

In this first approach, we propose a construction of the polynomial p such that $p(\beta)=0$ using specific $c(\alpha)$. The proposition below summarizes the main idea of this section.

Proposition 1. *Let m,r be two integers and p be an irreducible factor of $R(X)=X^{mr}+\sum_{i=0}^{m-1} z_i(X^m)X^i$, where $\deg z_i(X)<r$. Then, in $\mathbb{F}_{2^n}=\mathbb{F}_2[X]/(p)$, the elements $\beta=X$ and $\alpha=\beta^m$ satisfy*

$$\beta^m=\alpha, \quad (18)$$

$$\alpha^r=\sum_{i=0}^{m-1} z_i(\alpha)\beta^i. \quad (19)$$

Proposition 2. *Let m,r be two integers and p be an irreducible factor of*

$$R(X)=(X^m+1)^r+\sum_{i=0}^{m-1} z_i(X^m+1)X^i,$$

where $\deg z_i(X)<r$. Then, in $\mathbb{F}_{2^n}=\mathbb{F}_2[X]/(p)$, the elements $\beta=X$ and $\alpha=\beta^m+1$ satisfy

$$\beta^m=\alpha+1 \text{ and } \alpha^r=\sum_{i=0}^{m-1} z_i(\alpha)\beta^i.$$

Proof. In Proposition 1, (18) is a direct consequence of the definition of α. For (19), we know that

$$R(\beta)=0 \mod p.$$

If we replace $z_i(\beta^m)$ by $z_i(\alpha)$, we get the required (19). Thus, Proposition 1 is proved.

The proof of Proposition 2 can be tackled in the same way. □

To have an efficient step 3 in Algorithm 1, we must consider an ADPS providing a sparse expansion of α^r. In this case, the matrix M is sparse, and its coefficients are small. In particular, under the additional condition $\deg_\alpha Z = 0$ (Z is defined in (7)), we note that the ADPS given by the previous proposition provides a very efficient reduction process; indeed, according to Lemma 1, the number of calls of the semireduction algorithm is equal to $\lceil \frac{r}{r-1} \rceil = 2$ ($\deg_\alpha c = 1$ in Propositions 1 and 2).

This construction requires a factorization of a polynomial $R(X)$. This computation is only done once a time, during the construction of the field and the ADPS. This factorization can be done efficiently using different algorithms depicted in [10, chap. 14]. These methods have a polynomial complexity in the degree [10, p. 380], and thus are efficient for quite big n.

We give here one example, where we construct an ADPS for $\mathbb{F}_{2^{19}}$ using Proposition 2.

Example 3. We consider here the finite field $\mathbb{F}_{2^{13}}$. For $m = 3$ and $r = 5$, we determine Z such that z_i is a constant equal to 0 or 1, and such that the polynomial $(X^m + 1)^r - \sum_{i=1}^{m-1} z_i X^i$ admits an irreducible factor $P_\beta(X)$ of degree 13

$$(X^3 + 1)^5 + X + 1 = (X^2 + X)(X^{13} + X^{12} + X^{11} + X + 1). \tag{20}$$

Hence, $\mathbb{F}_{2^{13}} = \mathbb{F}_2[X]/(P_\beta(X))$ is defined. According to the fact that β verifies $(\beta^3 + 1)^5 = \beta^2 + 1$, and that $\alpha = \beta^3 + 1$, by the system $\mathcal{S} = (\alpha, 5, \beta, 3, P_\beta)$ such that:

$$\begin{aligned} &\beta^3 = \alpha + 1, \\ &\alpha^5 = \beta + 1 \quad (\text{i.e.}, \deg_\alpha Z = 0). \end{aligned}$$

♢

4.2 Construction of the Minimal Polynomial of α

For a more general $c(\alpha)$ (always with a very small degree), we have not been able to find a similar construction to Proposition 1. Thus, we propose to construct the irreducible polynomial p in the case $\alpha = X$ and the following equations hold:

$$\beta^m = c(\alpha), \tag{21}$$

$$\alpha^r = \sum_{i=0}^{m-1} z_i(\alpha)\beta^i. \tag{22}$$

If we multiply successively (22) by β^i for $i = 0, \ldots, m-1$ and reduce it relatively to β using (21), we obtain the following equations:

$$\begin{aligned} (\alpha^r + z_0(\alpha)) + z_1(\alpha)\beta + \cdots \\ \cdots + z_{m-1}(\alpha)\beta^{m-1} = 0, \\ c(\alpha)z_{m-1}(\alpha) + (\alpha^r + z_0(\alpha))\beta + z_1(\alpha)\beta^2 + \cdots \\ \cdots + z_{m-2}(\alpha)\beta^{m-1} = 0, \\ c(\alpha)z_{m-2}(\alpha) + c(\alpha)z_{m-1}(\alpha)\beta + \cdots \\ \cdots + z_{m-3}(\alpha)\beta^{m-1} = 0, \\ \vdots \qquad\qquad \vdots \\ c(\alpha)z_1(\alpha) + c(\alpha)z_2(\alpha)\beta + \cdots \\ \cdots + c(\alpha)z_{m-1}(\alpha)\beta^{m-2} + (\alpha^r + z_0(\alpha))\beta^{m-1} = 0. \end{aligned} \tag{23}$$

We note that any linear combination over $\mathbb{F}_2[\alpha]$ of the above equations is equal to zero. In other words, if we define the matrix $M(X)$ with coefficients in $\mathbb{F}_2[X]$ as follows:

$$M(X) = \begin{bmatrix} z_0(X) & c(X)z_{m-1}(X) & \cdots & c(X)z_1(X) \\ z_1(X) & z_0(X) & \cdots & c(X)z_2(X) \\ \vdots & & & \vdots \\ z_{m-1}(X) & z_{m-2}(X) & \cdots & z_0(X) \end{bmatrix}, \tag{24}$$

and then, if I denotes the $m \times m$ identity matrix, for every $U = (u_0(\alpha), \ldots, u_{m-1}(\alpha))$ with $u_i(\alpha) \in \mathbb{F}_2[\alpha]$, we have $(\alpha^r I - M(\alpha)) \cdot U = 0$. This implies that α is a root of

$$\det(X^r I - M(X)) = 0. \tag{25}$$

The polynomial p can thus be taken as a factor of $\det(X^r I - M(X))$. Knowing p, we construct $\mathbb{F}_{2^n}$ as $\mathbb{F}_{2^n} = \mathbb{F}_2[X]/(p)$ and $\alpha = X$. Since β is a root of the two polynomials $Y^m + c(\alpha)$ and $\alpha^r + \sum_{i=0}^{m-1} z_i(\alpha)Y^i$ of $\mathbb{F}_{2^n}[Y]$, we find the expression of β in $\mathbb{F}_{2^n}$ by computing

$$\gcd(Y^m + c(\alpha), \alpha^r + \sum_{i=0}^{r-1} z_i(\alpha)Y^i),$$

which must have $Y - \beta$ as factor.

Proposition 3. *If we consider $\alpha = X$ and β a root of $\gcd(Y^m + c(\alpha), \alpha^r + \sum_{i=0}^{r-1} z_i(\alpha)Y^i)$, with a polynomial p factor of $\det(X^r I - M(X))$, where $M(X)$ is defined as in (24), then, α and β satisfy:*

$$\beta^m = c(\alpha), \tag{26}$$

$$\alpha^r = \sum_{i=0}^{m-1} z_i(\alpha)\beta^i, \tag{27}$$

and they define an ADPS over $\mathbb{F}_2[X]/(p(X))$.

Example 4. We consider here the case $m = 5, r = 4$ and we look for a field $\mathbb{F}_{2^n}$ and two elements $\alpha, \beta \in \mathbb{F}_{2^n}$ such that

$$\beta^5 = \alpha^2 + \alpha + 1 \quad \text{and} \quad \alpha^4 = \beta^4 + \beta^3 + \beta^2.$$

As explained above, we first compute the matrix $M(X)$

$$M(X) = \begin{bmatrix} 1 & 0 & 0 & X^2 + X + 1 & X^2 + X + 1 \\ 1 & 1 & 0 & 0 & X^2 + X + 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Now we find $p(X) = X^{13} + X^{12} + X^8 + X^7 + X^6 + X^3 + 1$ such that:

$$p(X) \mid \det(X^r + M(X)).$$

Then, we can choose $n = 13$, with $\mathbb{F}_{2^{13}} = \mathbb{F}_2[X]/(p)$ and $\alpha = X$.

We compute β by determining the great common divisor: $\gcd(Y^m + \alpha^2 + \alpha + 1, \alpha^4 + Y^2 + Y^3 + Y^2) = (Y - \beta)$. This gives $\beta = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7 + \alpha^9$. ♢

Remark 4. We conjecture that it is possible for every binary field to find an ADPS providing an efficient reduction process, and thus, an efficient multiplication.

Remark 5. Proposition 3 can easily be extended for $\alpha \neq X$.

5 Improved ADPS Multiplier Using Lagrange Representation

In Section 3, we gave a general form for ADPS multiplication. In this section, we study a modified version of Algorithm 1, using the FFT/Lagrange approach presented in [11]. It is based on the following remark: let $\phi(\alpha)$ be a polynomial satisfying $\deg_\alpha \phi(\alpha) > (2r-2) + \deg_\alpha c(\alpha)$; the two first steps of Algorithm 1 can be done through

$$B \leftarrow UV \mod (\beta^m - c(\alpha), \phi(\alpha)).$$

Performing the operations modulo $(\beta^m - c(\alpha), \phi(\alpha))$ means that we reduce UV in β modulo $\beta^m - c(\alpha)$ and that we reduce the result in α modulo $\phi(\alpha)$. Indeed, in Algorithm 1, we have $\deg_\alpha(UV \bmod (\beta^m - c(\alpha)) \leq 2r - 2 + \deg_\alpha c(\alpha)$ (see formula 6); thus, if we reduce $UV \bmod (\beta^m - c(\alpha)$ modulo $\phi(\alpha)$, we do not change it. If we denote $\mathcal{R} = \mathbb{F}_2[\alpha]/(\phi(\alpha))$, the product UV is a product of polynomials in $\mathcal{R}[\beta]$ modulo $\beta^m - c(\alpha)$. The strategy in [11] was to choose $\phi(\alpha)$ such that this product is easy to compute. We first state some background on Lagrange Representation.

5.1 Lagrange Representation

Let $\mathcal{R}$ be a ring and $\mathcal{R}[\beta]$ be the polynomial ring over $\mathcal{R}$. The Lagrange representation of a polynomial of degree $(m-1)$ in $\mathcal{R}[\beta]$ is given by its values at m distinct points [17]. For us, these m points will be the roots $\zeta_i \in \mathcal{R}$ of a polynomial $E = \prod_{i=0}^{m-1}(\beta - \zeta_i) \in \mathcal{R}[\beta]$. From an arithmetic point of view, this is related to the Chinese Remainder Theorem which asserts that the following application is an isomorphism:

$$\begin{aligned} \mathcal{R}[\beta]/(E(\beta)) &\xrightarrow{\sim} \prod_{i=0}^{m-1} \mathcal{R}[\beta]/(\beta - \zeta_i), \\ A &\mapsto (A \mod (\beta - \zeta_i))_{i \in \{0,\ldots,m-1\}}. \end{aligned} \tag{28}$$

The computation of $A \bmod (\beta - \zeta_i)$ is simply the computation of $A(\zeta_i)$. In other words, the image of $A(\beta)$ by the isomorphism (28) is nothing else than the multipoints evaluation of A at the roots of E. This fact motivates the following Lagrange representation of the polynomials:

Definition 4 (Lagrange representation [17]). *Let $A \in \mathcal{R}[\beta]$ with $\deg A < m$, and $\zeta_0, \ldots, \zeta_{m-1}$ be the m distinct roots of a polynomial $E(\beta)$*

$$E(\beta) = \prod_{i=0}^{m-1}(\beta - \zeta_i).$$

If $a_i = A(\zeta_i)$ for $0 \leq i \leq m-1$, the Lagrange representation (LR) of $A(\beta)$ is defined by $\bar{A} = (a_0, \ldots, a_{m-1})$.

Lagrange representation is advantageous to perform operations modulo E; this is a consequence of the Chinese Remainder Theorem. Specifically, the arithmetic modulo E in classical polynomial representation can be costly if E has a high degree. In LR, the arithmetic is decomposed into m independent arithmetic units, with each unit performing arithmetic modulo a very simple polynomial $(\beta - \zeta_i)$. Furthermore, arithmetic modulo $(\beta - \zeta_i)$ is the arithmetic in $\mathcal{R}$ since the product of two zero-degree polynomials is just the product of the two constant coefficients.

5.2 Multiplication Using Lagrange Representation

Let us see how to use Lagrange representation to perform the product

$$U(\beta)V(\beta) \mod (\beta^m - c(\alpha), \phi(\alpha)).$$

This is the case if the polynomial $E(\beta) = \beta^m - c(\alpha)$ splits modulo $\phi(\alpha)$:

$$E(\beta) = \prod_{i=0}^{m-1}(\beta - \zeta_i) \bmod \phi(\alpha).$$

We obtain Algorithm 4 which results from this previous remark.

Algorithm 4. ADPS-LR Multiplication.
Require: U, V expressed trough an ADPS $\mathcal{B} = (\alpha, r, \beta, m, p)$.
Ensure: R in $\mathcal{B}$ such that $R = UV$ in $\mathbb{F}_{2^n}$
$\bar{U} \leftarrow Convert_{ADPS \to LR}(U)$
$\bar{V} \leftarrow Convert_{ADPS \to LR}(V)$
$\bar{B} \leftarrow \bar{U} \times \bar{V}$
$B \leftarrow Convert_{LR \to ADPS}(\bar{B})$
$W \leftarrow CR(B)$, (Algorithm 3)

The first two steps consist in computing the Lagrange representation of U and V from their ADPS representation. These two operations can be done in parallel.

The operations to compute $\bar{B}$ are performed in Lagrange representation, and then, can be easily parallelized as m independent multiplications in $\mathbb{F}_2[\alpha]/(\phi(\alpha))$. The operation $Convert_{LR \to ADPS}(\bar{B})$ refers to conversion from Lagrange representation to ADPS representation. The resulting B is thus equal to $U \times V \bmod \beta^n - c(\alpha)$. To get W, we have to just apply the coefficient reduction process (Algorithm 3).

We thus need to perform the conversions LR $\leftrightarrow$ ADPS efficiently.

5.3 Conversion LR $\leftrightarrow$ ADPS

An efficient implementation of conversions between Lagrange representations modulo $\phi(\alpha)$ and ADPS representation relies on the binomial form of $E(\beta) = \beta^m - c(\alpha)$. As stated in the following lemma, in this situation the roots of E have a special form:

Lemma 3. *Let $\mathcal{R} = \mathbb{F}_2[\alpha]/(\phi(\alpha))$ be such that $\phi(\alpha)$ is irreducible (i.e., $\mathcal{R}$ is a field) and let $E = \beta^m - c(\alpha)$ be a binomial polynomial which splits totally in $\mathcal{R}[\beta]$*

$$E(\beta) = \prod_{i=0}^{m-1}(\beta - \zeta_i), \quad \zeta_i \in \mathcal{R},$$

and such that the ζ_i are pairwise distinct. Then, there exists $\omega \in \mathcal{R}$, a primitive mth root of unity, and an element $\mu \in \mathcal{R}$ such that after reordering the ζ_i

$$\zeta_i = \mu\omega^i.$$

Proof. We fix $\mu = \zeta_0$ (i.e., μ is a root of E, as every ζ_i). Since, $\mathcal{R}$ is a field, μ^{-1} exists. We claim that the m distinct elements ζ_i/μ are m roots of unity. Indeed, we get:

$$(\zeta_i/\mu)^m = (\zeta_i)^m/\mu^m = c(\alpha)/c(\alpha) = 1,$$

since μ and ζ_i are roots of E. Moreover, since there are m distinct roots of unity in $\mathcal{R}$ and $\phi(\alpha)$ is irreducible, one of these roots must be a primitive mth root of unity. We call it ω. We can reorder the ζ_i to get $\zeta_i/\mu = \omega^i$ which gives $\zeta_i = \mu\omega^i$ as announced in the lemma. □

Using this form of the roots of E, we can perform the multipoint evaluation of the polynomial $A(\beta)$ in ζ_i (which corresponds to compute $\bar{A}$, the Lagrange representation of A) as follows:

1. set $\widetilde{A}(\beta) = A(\mu\beta) = \sum_{i=0}^{m-1} a_i\mu^i\beta^i$,
2. compute $\bar{A} = DFT_\phi(\widetilde{A}, m, \omega)$,

where $DFT_\phi(\widetilde{A}, m, \omega)$ is the evaluation of the polynomial $\widetilde{A}$ in the mth roots of unity ω^i for $i = 0, \ldots, m-1$. Similarly, the Lagrange interpolation, which computes $A(\beta)$ from $\bar{A}$, can be done by reversing the previous process.

Hence, the operations $Convert_{Pol\to LR}$ and $Convert_{LR\to Pol}$ have both a cost of m multiplications modulo ϕ and one Discrete Fourier Transform. This last operation can be done efficiently by using FFT algorithm [9, Section 8.2].

5.4 Hardware Architecture for FFT

We present an architecture to perform the FFT computation of a polynomial $A(\beta) \in \mathcal{R}[\beta]$ of degree $(m-1)$, keeping in mind our targeted Lagrange conversion. Note that the FFT process needs to be performed using the ternary method, since the binary one is not feasible over characteristic two rings [22]. Thus, in this section, we focus on the ring $\mathcal{R} = \mathbb{F}_2[\alpha]/(\phi(\alpha))$, where $\phi(\alpha) = \alpha^{2\nu/3} + \alpha^{\nu/3} + 1$, ν is a multiple of m, and $m = 3^s$. Hence, we have $\nu = \gamma m$.

Remark 6. We remind that one condition on $\phi(\alpha)$ is that its degree is greater than or equal to $2r - 1 + \deg c(\alpha)$ (formula 6). Thus, when $\deg\phi(\alpha) = 2\nu/3$, this means that r satisfies

$$r \le \nu/3 + 1 - \deg c(\alpha). \tag{29}$$

For efficiency reasons, r should be close to this upper bound.

Let $\omega = \alpha^\gamma$ be a primitive mth root of unity[2] in $\mathbb{F}_2[\alpha]/(\phi(\alpha))$ and let $\theta = \omega^{m/3}$ be a third root of unity. The ternary FFT process is based on the following three-way splitting of A:

$$A_1 = \sum_{j=0}^{m/3-1} a_{3j}\beta^{3j},$$

$$A_2 = \sum_{j=0}^{m/3-1} a_{3j+1}\beta^{3j},$$

$$A_3 = \sum_{j=0}^{m/3-1} a_{3j+2}\beta^{3j},$$

such that $A = A_1 + \beta A_2 + \beta^2 A_3$.

2. We note that $\phi(\alpha)(\alpha^{\nu/3}+1) = \alpha^\nu + 1 = 0$ over $\mathbb{F}_2[\alpha]/(\phi(\alpha))$.

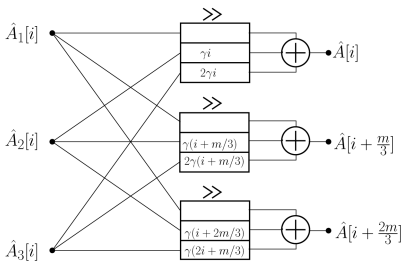

Fig. 1. Ternary butterfly operator.

Let $\widehat{A}[i] = A(\omega^i)$ be the ith coefficient of $DFT_\phi(A, m, \omega)$. Let us also denote by $\hat{A}_1[i] = A_1(\omega^{3i})$, $\hat{A}_2[i] = A_2(\omega^{3i})$, and $\hat{A}_3[i] = A_3(\omega^{3i})$ the coefficients of the DFT of order $m/3$ of, respectively, A_1, A_2, and A_3 (remind that ω^3 is an $m/3$ root of unity).

The following relations can be obtained by evaluating $A = A_1 + \beta A_2 + \beta^2 A_3$ in $\omega^i, \omega^{i+m/3}$, and $\omega^{i+2m/3}$:

$$\begin{aligned}
\hat{A}[i] &= \hat{A}_1[i] + \omega^i\hat{A}_2[i] + \omega^{2i}\hat{A}_3[i],\\
\hat{A}[i+m/3] &= \hat{A}_1[i] + \theta\omega^i\hat{A}_2[i] + \theta^2\omega^{2i}\hat{A}_3[i],\\
\hat{A}[i+2m/3] &= \hat{A}_1[i] + \theta^2\omega^i\hat{A}_2[i] + \theta\omega^{2i}\hat{A}_3[i].
\end{aligned} \tag{30}$$

This operation is frequently called the butterfly operation. It can be performed efficiently, if we compute modulo $\phi(\alpha)(\alpha^{\nu/3}+1) = \alpha^\nu + 1$ instead of $\phi(\alpha)$. Indeed, in this case, $\omega = \alpha^\gamma$ and a multiplication $a(\alpha) \times \omega^i$ modulo $\alpha^\nu + 1$ is a simple cyclic shift. The butterfly circuit (Fig. 1) is a consequence of this remark and of the relations given in (30).

In Fig. 1, the blocks noted $\gg$ refer to a simple shift operation by the given value and the $\bigoplus$ blocks refer to XOR operator. When no value is given, then shift operation is not performed.

Within the FFT, the computations of $\hat{A}_1, \hat{A}_2$, and $\hat{A}_3$ are done recursively in the same way. These polynomials are split in three parts and butterfly operations are applied again. This process is done recursively until constant polynomials are reached.

If we entirely develop this recursive process, we obtain the schematized architecture in Fig. 2.

Let us now evaluate the complexity of this architecture. It is composed of $\log_3(m)$ stages, where each stage consists of m operations in a butterfly way. Each of these operations requires 2ν XOR gates, and has a delay of $2T_X$, where T_X is the delay of one XOR gate. The final reduction of the coefficients modulo $\phi(\alpha)$ requires $\frac{2}{3}\nu m$ XOR for a delay of T_X. Consequently, this architecture has a space complexity of

$$\mathcal{S}(FFT_{\phi(\alpha)}) = \left(2\nu m\log_3(m) + \frac{2}{3}\nu m\right)\text{XOR} \tag{31}$$

and a delay of

$$\mathcal{D}(FFT_{\phi(\alpha)}) = (2\log_3(m) + 1)T_X. \tag{32}$$

6 Architecture and Complexity

We now present a parallel architecture associated to Algorithm 4 in the special case where $\phi(\alpha) = \alpha^{2\nu/3} + \alpha^{\nu/3} + 1$. This choice allows us to use the FFT circuit

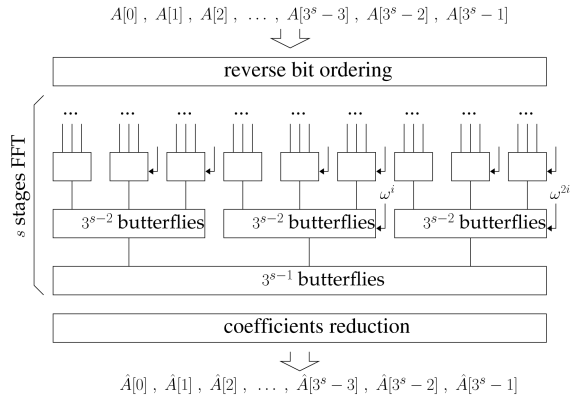

Fig. 2. Ternary FFT circuit.

presented in the previous section. The architecture of our binary field multiplier is given in Fig. 3. It is constituted of FFT blocks, multipliers modulo $\phi(\alpha)$ (referenced by $Mult_\phi$) and coefficient reduction block (referenced by $CoeffRed$). As much as possible, all the computations are parallelized.

6.1 Complexity Evaluation of Each Block

We now evaluate the complexity of this architecture block by block.

- *Complexity of $Mult_\phi$ blocks.* Since we consider $\phi(\alpha) = \alpha^{2\nu/3} + \alpha^{\nu/3} + 1$ and $\nu = \gamma m = \gamma 3^s$, if ν is a power of three, then we can use the multiplier of Fan and Hasan [6] to perform multiplication modulo ϕ. The complexity (cf. Table 1) of these blocks is easily deduced from [6, Table 1].
- *Complexity of FFT blocks.* The FFT blocks are designed using the ternary method presented in the previous section. Therefore, their complexities are given in (31) and (32).
- *Complexity of $CoeffRed$ blocks.* Recall that the coefficient reduction is performed (Algorithm 3) by computing

$$W \leftarrow SR(Q)\alpha^t + R$$

N times, where (see Lemma 1)

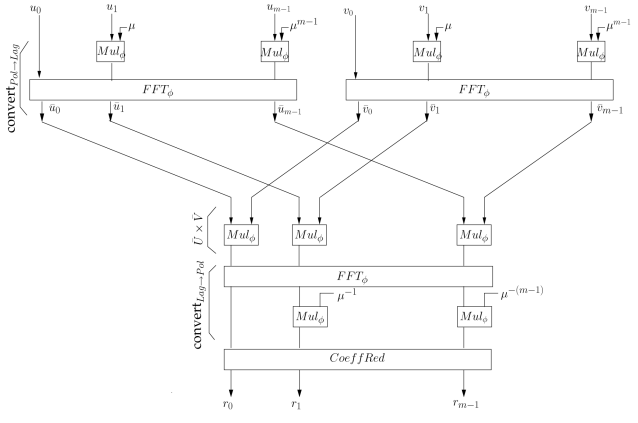

Fig. 3. DPS-Lagrange Multiplier.

TABLE 1
Complexity of Multipliers Modulo ϕ

	Mul_ϕ	
	Space	**Time**
#AND	$0.52\nu^{\log_3(6)}$	1
#XOR	$2.5\nu^{\log_3(6)} - 8\nu/3 - 4/5$	$3\log_3(2\nu/3) + 1$

$$N \leq \left\lceil \frac{r - 1 + \deg_\alpha c(\alpha)}{r - \deg_\alpha Z - \deg_\alpha c(\alpha)} \right\rceil.$$

At each time, to compute W we need only to compute $SR(Q)$, and no more operation is needed since $\deg_\alpha(SR(Q)\alpha^t) \geq t$ and $\deg_\alpha(R) < t$. Consequently, the corresponding architecture of the coefficient reduction consists of N circuits performing a semireduction.

The semireduction process (Algorithm 2) consists of the following operation:

$$B \leftarrow \underline{B} + M \cdot \overline{B},$$

where B is such that $\deg B \leq r + \Delta$ and $\underline{B} < r$ and $\overline{B} < \Delta$. The matrix M is as follows:

$$\begin{bmatrix} z_0(\alpha) & c(\alpha)z_{m-1}(\alpha) & \cdots & c(\alpha)z_1(\alpha) \\ z_1(\alpha) & z_0(\alpha) & \cdots & c(\alpha)z_2(\alpha) \\ \vdots & & & \vdots \\ z_{m-1}(\alpha) & z_{m-2}(\alpha) & \cdots & z_0(\alpha) \end{bmatrix}.$$

The complexity of the semireduction is thus related to Z, the ADPS representation of α^r, and to $c(\alpha)$ (cf. Section 3.2). We assume here that Z is sparse as polynomial in β and has degree 0 in α. We also assume that $\deg_\alpha c(\alpha)$ is small relatively to r. Consequently, in the matrix M, the z_i are equal to 0 or 1 and the matrix coefficients are equal to 1, to $c(\alpha)$, or to 0. Thus, for computing the matrix vector product $M \cdot \overline{B}$, we first compute $c(\alpha)\overline{b}_i(\alpha)$ for $i = 1, \ldots, m-1$, then, for each row of the matrix M, we add at most $HW(Z)$ coefficients of $\{c(\alpha)\overline{b}_i(\alpha), i = 1, \ldots, m-1\} \cup \{\overline{b}_i(\alpha), i = 0, \ldots, m-1\}$, where $HW(Z)$ is the number of $z_i(\alpha) \neq 0$. We obtain the complexity of the resulting architecture in Table 2.

The architecture for the multiplication by $c(\alpha)$ corresponds to the parallel binary tree of XOR, and no AND gates are needed since $c(\alpha)$ is a constant. We, finally, can deduce the complexity of the CoeffRed architecture, it is just N times the complexity of the semireduction architecture.

TABLE 2
SR Complexity

Operation	Complexity	
	Space (# XOR)	Time
$m-1$ mult. by $c(\alpha)$	$(m-1)r(\deg_\alpha c(\alpha) - 1)$	$\log_2(\deg_\alpha(c(\alpha)))T_X$
$m(HW(Z)-1)$ add. in $M \cdot \overline{B}$	$mr(HW(Z)-1)$	$\log_2(HW(Z))T_X$
m coeff. additions in $\underline{B} + \overline{B} \cdot M$	mr	$1T_X$
Overall Complexity	$mrHW(Z)+$ $(m-1)r(\deg c(\alpha) - 1)$	$\log_2(\deg_\alpha(c(\alpha)))T_X+$ $(\log_2(HW(Z)+1)T_X$

TABLE 3
Complexity Comparison of Multiplication in $GF(2^n)$

Method	Space Complexity		Time Complexity	
	# AND	# XOR	T_A	T_X
This paper	$8.82n^{1.31} - 3.82n^{0.82}$	$42.4n^{1.31} - 18.4n^{0.82}$ $-37.5n^{1/3} + 1.38n^{1/2} + 2.4 + 27n + 9n\log_3(n)$	3	$(13\log_3(n) + 9\log_3(2)) + 6$
GNP [11]	$14.5n^{1.31}$	$69.6n^{1.31} - 31n + n^{0.5}(8\log_3(n) + 39)$	8	$16\log_3(n) + 20$
FH^* binary	$n^{1.58}$	$5.5n^{1.58} - 5n - 0.5$	1	$2\log_2(n) + 1$
FH^* ternary	$n^{1.63}$	$4.8n^{1.63} - 4n - 0.8$	1	$3\log_3(n) + 1$

$FH^* =$ *[6], where for binary* $n = 2^t$, *for ternary* $n = 3^t$.

6.2 Overall Complexity of the Multiplier

We deduce the overall complexity of our multiplier (Table 4). We first give the number of operating blocks. Their corresponding space complexity is denoted by $\mathcal{S}$, and their time complexity is denoted by $\mathcal{D}$. Thus, the space complexity is given by:

$$(4m-3)\mathcal{S}(Mul_\phi) + 3\mathcal{S}(FFT_\phi) + \mathcal{S}(CoeffRed).$$

Similarly, the critical path of this architecture gives the delay of our multiplier:

$$3\mathcal{D}(Mul_\phi) + 2\mathcal{D}(FFT_\phi) + \mathcal{D}(CoeffRed).$$

With the previous expression of the complexity of FFT block ((31) and (32)), RedCoeff (Table 2), and $Mult_\phi$ (Table 1), we find the space complexity in terms of the number of XOR and AND gates.

6.2.1 Asymptotic Complexity in n

In this part, we consider that $\nu = m$ which is a correct asymptotic assumption that simplifies the formulations. In order to construct a DPS-FFT multiplier, we must have $n \leq rm$ and $r \leq m/3$, where n is the degree of the field $\mathbb{F}_{2^n}$. This implies that $n \leq m^2/3$, and the best n are such that $m \cong \sqrt{3n}$. In Table 3, we give the complexity for this case (we suppose that $HW(Z) \leq 3$ and $\deg_\alpha c(\alpha) \leq 6$ and $N \leq 2$, which corresponds to practical situations).

We also give in Table 3 the complexity of the best known method, regarding space and time complexity, to perform binary field multiplication. Specifically, we give the complexity of the multiplier of [11], which has asymptotically the smaller space complexity. We also give the complexity of [6], which is better than [11] for smaller field. We do not give the complexity of [23] and [1], which also present subquadratic space complexity multipliers, since their complexities are worse than [6].

We can remark that our approach has a space complexity with the same order as [11], i.e., $O(n^{1.31})$. But we improve by 33 percent. Our multiplier is also 18 percent faster than the multiplier of [11].

The multiplier of Fan and Hasan has space complexity with order $O(n^{1.56})$ but it is, in general, faster than our multiplier. But, the Fan-Hasan approach is available only when n is a power of two or three. Our method is more general and is available for all n.

In Table 5, we give for different field sizes, a corresponding ADPS. We constructed such ADPS using the method of Section 4.2. These ADPSs admit an FFT multiplier. We do not give the polynomial $p(X)$, which defines the field, but it can be recovered by factoring the determinant given in (25). In each field, α satisfies $p(\alpha) = 0$.

In this table, we also give the corresponding complexity of DPS-FFT multiplier and Fan-Hasan multiplier using the formulas of Table 3. We can see that our multiplier becomes better than the Fan-Hasan multiplier around 3,000 bits. This is due to the constant factor in $n^{1.31}$. We could get better complexity if we could improve the multiplication by the constant μ^i and μ^{-i}. We point out, even if the given examples do not show it, that our method is available for field size recommended by NIST [3].

TABLE 4
Complexity of DPS-FFT Architecture

#AND	$0.52(4m-3)\nu^{\log_3(6)}$
#XOR	$(10m - 7.5)\nu^{\log_3(6)} - 26/3\nu m - 16/5m + 8/3\nu + 12/5$ $+6\nu m\log_3(m) + NmrHW(Z) + N(m-1)r(\deg c(\alpha))$.
Time	$(13\log_3(m) - 4 + 9\log_3(2))T_X$ $+(\log_2(HW(Z)) + \log_2(\deg c(\alpha)))NT_X + 3T_A$

7 Other Operations

For a practical use of ADPS multiplication, some additional operations could be necessary like conversion to classical polynomial systems, testing the equality of two elements, squaring, inversion, etc. We present here several methods to perform these operations.

7.1 Squaring

Let U be an element expressed in an ADPS system $\mathcal{S} = (\alpha, r, \beta, m, c, p)$ as $U = \sum_{i=0}^{m-1}\sum_{j=0}^{r-1} u_{i,j}\alpha^j\beta^i \pmod p$, with $u_{i,j} \in \{0,1\}$. To compute $A = U^2$, we use the well-known property which states that the squaring of polynomial in $\mathbb{F}_2[\beta,\alpha]$ consists in just multiplying the exponents by two

$$U^2 = \sum_{i=0}^{m-1}\sum_{j=0}^{r-1} u_{i,j}\alpha^{2j}\beta^{2i} \pmod p.$$

This computation is free of computation. After that, we just have to perform a reduction modulo $\beta^n - c(\alpha)$, and then, a reduction of the coefficients to have the ADPS representation of U^2.

Algorithm 5. ADPS squaring
Require: One ADPS $(\alpha, r, \beta, m, c, p)$ and one element of $\mathbb{F}_{2^n}$, $U = (u_0(\alpha), \ldots, u_{m-1}(\alpha))$
Ensure: W.
Polynomial squaring in $(\mathbb{F}_2[\alpha])[\beta]$.
$A(\beta) \leftarrow \sum_{i=0}^{m-1}\sum_{j=0}^{r-1} u_{i,j}\alpha^{2j}\beta^{2i}$
Polynomial reduction. $B(\beta) \leftarrow A(\beta) \bmod (\beta^m - c(\alpha))$
Coefficients reduction. $W(\beta) \leftarrow CR(B(\beta))$

TABLE 5
LR-ADPS Examples

Degree	$method$	System	Complexity		
			$\#AND$	$\#XOR$	$Delay$
241	$thispaper$	$r=9, m=27, c(\alpha)=\alpha^4+\alpha^2+\alpha+1, Z=\beta^{15}+\beta^2+1, \phi(\alpha)=\alpha^{54}+\alpha^{27}+1.$	70761	364074	$3T_A+47T_X$
241	FH	$ShiftedPolynomialBasis$	7736	36162	T_A+16T_X
569	$thispaper$	$r=7, m=81, c(\alpha)=\alpha^6+\alpha^5+\alpha^3+\alpha^2+\alpha, Z=\beta^{12}+\beta^2+1, \phi(\alpha)=\alpha^{54}+\alpha^{27}+1.$	216328	1.15×10^6	$64T_X+3T_A$
569	FH	$ShiftedPolynomialBasis$	46370	219663	$19TX+TA$
2753	$thispaper$	$r=34, m=81, c(\alpha)=\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha, Z=\beta^{25}+\beta^{14}+1, \phi(\alpha)=\alpha^{162}+\alpha^{81}+1.$	1.29×10^6	6.59×10^6	$3T_A+64T_X$
2753	FH	$ShiftedPolynomialBasis$	1.66×10^6	7.97×10^6	T_A+25T_X
6073	$thispaper$	$r=75, m=81 c(\alpha)=\alpha^6+\alpha^5+\alpha^4+\alpha^3+\alpha, Z=\beta^7+1, \phi(\alpha)=\alpha^{162}+\alpha^{81}+1.$	1.29×10^6	6.63×10^6	$3T_A+62T_X$
6073	FH	$ShiftedPolynomialBasis$	1.66×10^6	7.97×10^6	T_A+25T_X

We can directly implement this algorithm in hardware in the special case of ADPS given in Section 4. The corresponding space complexity is equal to the space complexity of the reduction modulo $\beta^m - c(\alpha)$ plus the complexity of RedCoeff.

- *Complexity of Polynomial reduction.* For the reduction modulo $\beta^m - c(\alpha)$, we have $m/2$ multiplications by $c(\alpha)$, and, at most $m/2$ additions of coefficients of degree $2r-2$. A multiplication by $c(\alpha)$ has a cost in space of $(\deg c(\alpha))(2r-1)$ XOR gates and a cost in time of $\log_2(\deg c(\alpha))T_X$.
- *Complexity of coefficient reduction.* We have already evaluated this cost in the previous section, and it is given in Table 2.

The resulting cost of the squarer is equal to

$$(Nmr(HW(Z)-1)+(N(m-1)r + m/2(2r-1)(\deg c(\alpha))+m/2(2r-1))$$

XOR for the space complexity and $N(\log_2(HW(Z))+(N+1)\log_2(\deg c(\alpha)+1)T_X$ in time. When $\deg_\alpha c(\alpha)$ and $HW(Z)$ are small, i.e., of order $O(1)$, the space complexity is $O(rm)$ XOR and the time complexity is equal to $O(1)T_X$.

7.2 Conversion between ADPS to Standard Polynomial

Let us represent an element expressed in an ADPS $\mathcal{S}$ as $U_{\mathcal{S}}=\sum_{i=0}^{m-1}\sum_{j=0}^{r-1}u_{i,j}\alpha^i\beta^j$. We get its standard polynomial representation by replacing each $\alpha^i\beta^j$ by their corresponding standard polynomial expression $\alpha(X)$ and $\beta(X)$

$$U=\sum_{i=0}^{n}u_{i,j}\alpha(X)^j\beta(X)^i \mod p.$$

We can perform this using a precomputed expression of $\alpha(X)^i\beta(X)^j \bmod p$ since these elements are constant. This strategy requires mr additions of degree n polynomial.

For the reverse conversion, i.e., from standard polynomial to ADPS, we use an ADPS representation of X

$$X_{\mathcal{S}}=\sum_{i=0}^{r-1}\sum_{j=0}^{m-1}x_{i,j}\alpha^i\beta^j.$$

Let $U(X)$ be an element of $\mathbb{F}_{2^n}$ in standard polynomial representation. We compute $U_{\mathcal{S}}$ by substituting X by $X_{\mathcal{S}}$ in $U(X)$. This method is developed in Algorithm 3.

The complexity of this method is equal to n calls of Algorithm 1.

Algorithm 6. Conversion from polynomial to ADPS
Require: A degree $n-1$ polynomial $U(X)$ in X.
Ensure: $U_{\mathcal{S}}(X)$ the representation of U in $\mathcal{S}$.
  $U_{\mathcal{S}} \leftarrow 0$
  for $i=n-1$ to 0 **do**
    $U_{\mathcal{S}} \leftarrow u_i + \text{ADPS\_Multiplication}(X_{\mathcal{S}}, U_{\mathcal{S}})$, (Algorithm 1)
  end for

7.3 Comparison of Elements

An ADPS is, in general, a redundant system. This means that a field element U can have different representations in this system. Consequently, the following question can occur during a computation: let $U_{\mathcal{S}}$ and $V_{\mathcal{S}}$ be two elements expressed in $\mathcal{S}$, are they equal? $U_{\mathcal{S}}$ and $V_{\mathcal{S}}$ could be two different DPS representations of one unique field element.

For equality, we consider a basis extracted from $\mathcal{S}$. Indeed, from linear algebra theory, we can extract a basis from each generating system of a vector space. Let

$$\mathcal{B}=\left\{\alpha^{i_1}\beta^{j_1},\ldots,\alpha^{i_n}\beta^{j_n}\right\}$$

be a basis extracted from an ADPS $\mathcal{S}$. In this basis, each element in $\mathbb{F}_{2^n}$ has a unique expression. Specifically, for $\alpha^i\beta^j$ such that $(i,j)\notin\{(i_1,j_1),\ldots,(i_n,j_n)\}$, we have

$$\alpha^i\beta^j=\sum_{k=1}^{n}\lambda_k^{(i,j)}\alpha^{i_k}\beta^{j_k}.$$

To get a representation of an element U in $\mathcal{B}$, we just have to replace $\alpha^i\beta^j$ by their corresponding expression in $\mathcal{B}$ in the DPS representation of U. Consequently, U is equal to V if the representation of $U-V$ in $\mathcal{B}$ is equal to 0.

7.4 Inversion

We do not know any method exploiting ADPS representation really efficiently. We mention here two methods. The first one uses the classical exponentiation method based on Fermat theorem. Using a square and multiply method, we compute $U^{-1}=U^{2^n-2}$.

The second method uses extended euclidean algorithm in standard polynomial multiplication. Specifically, if we have conversion operator, one can compute inversion of an element U as follows:

- Convert U expressed in ADPS to standard polynomial $U(X)$.
- Compute the inverse U^{-1} of U modulo $p(X)$ using the extended euclidean algorithm.
- Convert U^{-1} to the ADPS.

8 Conclusion

We have presented in this paper a new multiplication algorithm in the double polynomial system presented in [11]. We use a different approach for coefficient reduction. Specifically, using a sparse ADPS representation of α^r, the coefficient reduction becomes really simple and efficient. We avoid one multiplication used in the Montgomery strategy of [11]. We also give some method to construct ADPS which admits a sparse α^r. We give also algorithm for other operations: e.g., squaring, comparison, etc.

We have presented an architecture for this algorithm using the Lagrange and FFT approach. The resulting architecture is better than the multiplier of [11] by 33 percent in space and 18 percent in time. ADPS offers an interesting alternative to other approaches (see Table 3), with a complexity close to best known methods without restriction on n. We compare our approach with the best known subquadratic multiplier for small field: the multiplier of Fan and Hasan [6]. We show that our multiplier is asymptotically better than Fan and Hasan, and in practical uses, this is true when $243 < n \leq 673$.

References

[1] J.-C. Bajard, L. Imbert, and G.A. Jullien, "Parallel Montgomery Multiplication in GF(2^k) Using Trinomial Residue Arithmetic," *Proc. IEEE Symp. Computer Arithmetic (ARITH '05),* pp. 164-171, 2005.

[2] J.-C. Bajard, L. Imbert, and T. Plantard, "Modular Number Systems: Beyong the Mersenne Family," *Proc. Int'l Workshop Selected Areas in Cryptography (SAC '04),* pp. 159-169, 2005.

[3] M. Brown, D. Hankerson, J. López, and A. Menezes, "Software Implementation of the NIST Elliptic Curves over Prime Fields," *Topics in Cryptology—CT-RSA,* pp. 250-265, Springer, 2001.

[4] C. Doche, "Redundant Trinomials for Finite Fields of Characteristic 2," *Proc. Australasian Conf. Information Security and Privacy (ACISP '05),* pp. 122-133, 2005.

[5] E. DeWin, A. Bosselaers, S. Vandenberghe, P. DeGersem, and J. Vandewalle, "A Fast Software Implementation for Arithmetic Operations in GF(2^n)," *Advances in Cryptology—Asiacrypt '96,* pp. 65-76, Springer, 1996.

[6] H. Fan and A. Hasan, "A New Approach to Subquadratic Space Complexity Parallel Multipliers for Extended Binary Fields," *IEEE Trans. Computers,* vol. 56, no. 2, pp. 224-233, Feb. 2007.

[7] H. Fan and M.A. Hasan, "Subquadratic Computational Complexity Schemes for Extended Binary Field Multiplication Using Optimal Normal Bases," *IEEE Trans. Computers,* vol. 56, no. 10, pp. 1435-1437, Oct. 2007.

[8] S.T.J. Fenn, M. Benaissa, and D. Taylor, "GF(2^m) Multiplication and Division Over the Dual Basis," *IEEE Trans. Computers,* vol. 45, no. 3, pp. 319-327, Mar. 1996.

[9] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra.* Cambridge Univ. Press, 1999.

[10] J. von zur Gathen and J. Gerhard, "Polynomial Factorization over $\mathbb{F}_2$," *Math. Computation,* vol. 71, no. 240, pp. 1677-1698, 2002.

[11] P. Giorgi, C. Negre, and T. Plantard, "Subquadratic Binary Field Multiplier in Double Polynomial System," *Proc. Int'l Conf. Security and Cryptography (SECRYPT '07),* 2007.

[12] J. Guajardjo and C. Paar, "Efficient Algorithms for Elliptic Curve Cryptosystems," *Advances in Cryptology—Crypto '97,* pp. 342-356, Springer, 1997.

[13] J. Guajardo, T. Gneysu, S.S. Kumara, C. Paar, and J. Pelzl, "Efficient Hardware Implementation of Finite Fields with Applications to Cryptography," *J. Acta Applicandae Mathematicae,* vol. 93, nos. 1-3, pp. 75-118, Sept. 2006.

[14] A. Halbutogullari and C.K. Koc, "Mastrovito Multiplier for General Irreducible Polynomials," *IEEE Trans. Computters,* vol. 49, no. 5, pp. 503-518, May 2000.

[15] D. Hankerson, J. López Hernandez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01),* 2001.

[16] J.-C. Bajard, C. Negre, and T. Plantard, "Double Polynomial Basis Representation for Binary Field Arithmetic," Technical Report 5, Team DALI/Lab. ELIAUS, Univ. of Perpignan, July 2005.

[17] J.-C. Bajard, L. Imbert, and C. Negre, "Arithmetic Operations in Finite Fields of Medium Prime Characteristic using Lagrange Representation," *IEEE Trans. Computers,* vol. 55, no. 9, pp. 1167-1177, Sept. 2006.

[18] J.L. Massey and J.K. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US Patent 4,587,627, May 1986.

[19] E.D. Mastrovito, "VLSI Architectures for Computations in Galois Fields," PhD thesis, Dept. of Electrical Eng., Linkoping Univ., 1991.

[20] P.L. Montgomery, "Modular Multiplication without Trial Division," *Math. Computation,* vol. 44, no. 170, pp. 519-521, Apr. 1985.

[21] R.C. Mullin, I.M. Onyszchuk, S.A. Vanstone, and R.M. Wilson, "Optimal Normal Bases in GF(p^n)," *Discrete Applied Math.,* vol. 22, no. 2, pp. 149-161, 1989.

[22] A. Schonhage, "Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2," *Acta Informatica,* vol. 7, pp. 395-398, 1977.

[23] B. Sunar, "A Generalized Method for Constructing Subquadratic Complexity GF(2^k) Multipliers," *IEEE Trans. Computers,* vol. 53, no. 9, pp. 1097-1105, Sept. 2004.

[24] L.A. Tawalbeh, A.F. Tenca, S. Park, and C.K. Koc, "An Efficient Hardware Architecture of a Scalable Elliptic Curve Crypto-Processor over GF(2^m)," *Proc. SPIE Conf.,* pp. 216-226, Aug. 2005.

[25] L.A. Tawalbeh and A.F. Tenca, "An Algorithm and Hardware Architecture for Integrated Modular Division and Multiplication in GF(p) and GF(2^n)," *Proc. IEEE Int'l Conf. Application-Specific Systems, Architectures and Processors (ASAP '04),* pp. 247-257, 2004.

Jean-Claude Bajard received the PhD degree in computer science from the École Normale Supérieure de Lyon (ENS), France, in 1993. He is currently a professor at the Université Paris 6, France, and a member of the LIP 6. He taught mathematics in high school from 1979 to 1990 and served as a research and teaching assistant at the ENS in 1993. From 1994 to 1999, he was an assistant professor at the Université de Provence, Marseille, France. From 1999 to 2009, he was a professor at the University Montpellier 2, and was a member of the ARITH Team of the LIRMM. His research interests include computer arithmetic and cryptography.

Christophe Negre received the MS degree in mathematics and the PhD degree in computer science from the Université Montpellier 2, France, in 2001 and 2004, respectively. Since September 2004 , he is an assistant professor in the DALI Team at the Université de Perpignan, France. His research interests include computer arithmetic and cryptography.

Thomas Plantard received the MS and PhD degrees in computer science from the Université de Bordeaux in 2002 and the Université Montpellier 2, France, in 2005, respectively. Since September 2006, he has a postdoctoral position at the University of Wollongong, Australia. His research interests include cryptography and lattice theory.